Package 'hptmUsage'

January 21, 2026

Title Usage Analysis of Histone Post-Translational Modifications

Version 1.0.0

Description Preprocessing, visualization, and analysis (e.g.,
differential usage) of histone post-translational modifications
(hPTMs). It builds on the 'msqrob2PTM' workflow to enable robust and
statistically sound analysis of hPTM data.

License GPL (>= 3)

URL `https://github.com/rualmey/hptmUsage`

BugReports `https://github.com/rualmey/hptmUsage/issues`

Depends QFeatures (>= 1.18.0),
R (>= 4.3)

Imports archive,
Biobase,
Biostrings,
dplyr,
ExploreModelMatrix,
forcats,
fs,
ggbreak,
ggExtra,
ggmsa,
ggplot2,
ggrepel,
hexbin,
janitor,
limma,
MatrixGenerics,
methods,
MsCoreUtils,
msqrob2,
plotly,
purrr,
quarto,

ragg,
readr,
rlang,
stringr,
S4Vectors,
SummarizedExperiment,
svglite,
tibble,
tidyr,
tidyselect,
UniProt.ws,
vctrs,
vsn,
stringi

Suggests bench,
covr,
mockery,
testthat (>= 3.0.0),
vcr,
withr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

SystemRequirements mafft, quarto

Collate 'calculateUsage.R'
'data.R'
'deconvolute.R'
'generateReport.R'
'hptmMSA.R'
'hptmUsage-package.R'
'matchHistones.R'
'normalizeUsage.R'
'processMods.R'
'readProgenesis.R'
'replaceColData.R'
'tagContaminants.R'
'usagePlot.R'

Contents

`aligned_histones` *Aligned Human Histones*

Description

A default call to `alignHistones()` retrieved all human histone variants from UniProt (reviewed entries only, "2025-07-28 08:32:03 UTC") and added them to the predefined (HistoneDB 2.0 (1)) MSA profile.

Usage

```
aligned_histones
```

Format

A `list` of three `AAStringSetList` objects, each containing five elements corresponding to the five histone families (H1, H2A, H2B, H3, H4):

unaligned `AAStringSetList` containing the unaligned histone sequences

msa `AAStringSetList` containing the aligned histone sequences after adding each sequence to the curated MSA profile of the corresponding family

msa_ref `AAStringSetList` containing the aligned reference sequences for each family

References

(1) Draizen, E. J.; Shaytan, A. K.; Mariño-Ramírez, L.; Talbert, P. B.; Landsman, D.; Panchenko, A. R. HistoneDB 2.0: A Histone Database with Variants—an Integrated Resource to Explore Histones and Their Variants. Database (Oxford) 2016, 2016, baw014. https://doi.org/10.1093/database/baw014.

Examples

```
# List all Histone H3 variants in the object
aligned_histones$unaligned$H3 |>
  names()
# Show the MSA of histone family H3
aligned_histones$msa$h3 |>
  print()
# Show the aligned reference sequences
aligned_histones$msa_ref |>
  purrr::walk(print)
```

`alignHistones` *Align Histone Sequences Using MAFFT*

Description

Histone sequences can be aligned to prevent location mismatches between corresponding hPTMs, e.g, H31S57 vs. H3CS56.

Usage

```
alignHistones(
  unaligned_histones = NULL,
  return_alignment = FALSE,
  output_path = "./out/msa/",
  overwrite = FALSE,
  use_profiles = TRUE,
  nondefault_refseq_names = NULL,
  return_only_original = TRUE,
  quiet = TRUE,
  num_cores = -1,
  ...
)
```

Arguments

`unaligned_histones`
: An optional `AAStringSetList` of histone sequences to align, by family. If `NULL`, a default call to `histonesFromUniprot()` is made, i.e., sequences from all five histone families (human and reviewed) are retrieved.

`return_alignment`, `output_path`, `overwrite`
: Return the alignment? In order:

- A `logical(1)` indicating whether to save the alignment to `.fasta` files.
- A `character(1)` specifying the directory to save alignment files.
- A `logical(1)` indicating whether to overwrite existing files at `output_path`

`use_profiles`	A `logical(1)` or `character()` indicating whether to add sequences to existing histone MSA profiles. If `TRUE`, uses the curated alignments from HistoneDB 2.0 (2). Can also be a `character()` specifying paths to custom, pre-aligned `.fasta` files.
`nondefault_refseq_names`	An optional `character()` of non-default reference sequence names. If `NULL`, H1.1 and canonical H2A/H2B/H3/H4 are used. Be aware that changing this can change hPTM location definition!
`return_only_original`	A `logical(1)` indicating whether to return only the sequences present in `unaligned_histones` in the MSA. Optional, only used in profile mode.
`quiet`	A `logical(1)` indicating whether to suppress MAFFT output.
`num_cores`	An `integer(1)` specifying the number of CPU cores to use during alignment. Optional, `-1` uses all cores.
`...`	Additional arguments passed to MAFFT, overriding any defaults.

Details

To ensure consistent and accurate alignment regardless of the underlying sequences, this alignment is by default added into a predefined MSA profile using the MAFFT `--add` functionality (1). This predefined MSA profile is by default the curated alignment of the corresponding histone family retrieved from HistoneDB 2.0 (2).

It is therefore highly recommended to leave these settings at default, so that hPTM definition will be consistent, even when the sequences to align change.

Value

A list containing the unaligned input sequences (`unaligned`), the multiple sequence alignment (`msa`), and optionally the aligned reference sequences (`msa_ref`) if `use_profiles` is not `FALSE`. Each list element is a named `AAStringSetList`. An error is raised if MAFFT fails alignment for any histone family.

References

(1) Katoh, K.; Standley, D. M. MAFFT Multiple Sequence Alignment Software Version 7: Improvements in Performance and Usability. Molecular Biology and Evolution 2013, 30 (4), 772–780. `https://doi.org/10.1093/molbev/mst010`.

(2) Draizen, E. J.; Shaytan, A. K.; Mariño-Ramírez, L.; Talbert, P. B.; Landsman, D.; Panchenko, A. R. HistoneDB 2.0: A Histone Database with Variants—an Integrated Resource to Explore Histones and Their Variants. Database (Oxford) 2016, 2016, baw014. `https://doi.org/10.1093/database/baw014`.

Examples

```
## Not run:
# Align default human histone sequences from UniProt (reviewed entries only)
aligned_human_histones <- alignHistones()
```

```
# Align a custom set of histone sequences
library(Biostrings)
h3_seqs <- AAStringSet(c(H3.1-Ntail = "ARTKQTARKSTGGKAPRKQLATKAARKSAPATGGVKKPH",
                         H3.3-Ntail = "ARTKQTARKSTGGKAPRKQLATKAARKSAPSTGGVKKPH"))
h4_seqs <- AAStringSet(c(H4-Ntail = "SGRGKGGKGLGKGGAKRHRKVLRDNIQGITKPAIRR"))
unaligned_seqs <- AAStringSetList(H3 = h3_seqs, H4 = h4_seqs)
aligned_custom_seqs <- alignHistones(unaligned_histones = unaligned_seqs)

# Align mouse histones instead of human
mouse_unaligned <- historesFromUniprot(query = c("organism_id:10090", "reviewed:true"))
aligned_mouse <- alignHistones(unaligned_histones = mouse_unaligned)

# Perform a de novo alignment instead of adding to a profile
denovo_aligned <- alignHistones(use_profiles = FALSE)

# Pass additional arguments to MAFFT, e.g., to change the gap opening penalty
alignHistones(op = 3.0)

# Use non-default reference sequences, note that this is not recommended!
alignHistones(
  nondefault_refseq_names = c(
    H1 = "H1.0|Bos|NP_001069955.1 Bos|NP_001069955.1|H1.0 Bos_H1.0_115496898",
    H2A = "H2A.1|Homo|NP_734466.1 Homo|NP_734466.1|H2A.1 Homo_H2A.1_25092737",
    H2B = "H2B.1|Canis|XP_005640164.1 Canis|XP_005640164.1|H2B.1 Canis_H2B.1_545554624",
   H3 = "H3.3|Arabidopsis|NP_195713.1 Arabidopsis|NP_195713.1|H3.3 Arabidopsis_H3.3_15236103",
   H4 = "canonical_H4|Mus|NP_291074.1 Mus|NP_291074.1|canonical_H4 Mus_canonical_H4_21361209"
  )
)

## End(Not run)
```

calculateUsage *Calculate hPTM/Variant Usage*

Description

hPTM or histone variant usage is calculated by first normalizing histone peptidoforms (i.e., subtracting the "parent protein" abundance) and then summarizing to the hPTM/variant level.

Usage

```
calculateUsage(
  object,
  ...,
  target = c("ptm", "variant"),
  usage_level = c("histone", "histone_family", "histone_group"),
  deconv = "mods_ref",
  sep = ";",
  group = "histone_group"
)

## S4 method for signature 'QFeatures'
calculateUsage(
  object,
  i,
  name = NULL,
  identifier = "feature_number",
  ...,
  target = c("ptm", "variant"),
  usage_level = c("histone", "histone_family", "histone_group"),
  deconv = "mods_ref",
  sep = ";",
  group = "histone_group"
)

## S4 method for signature 'SummarizedExperiment'
calculateUsage(
  object,
  ...,
  target = c("ptm", "variant"),
  usage_level = c("histone", "histone_family", "histone_group"),
  deconv = "mods_ref",
  sep = ";",
  group = "histone_group"
)
```

Arguments

`object`	The `QFeatures` or `SummarizedExperiment` object from which usage needs to be calculated.
`...`	Additional arguments passed to `QFeatures::aggregateFeatures()`.
`target`	A `character(1)` defining whether PTM or histone variant usage will be calculated.
`usage_level`	A `character(1)` defining the level at which usage will be calculated, i.e., what is considered the "parent protein". Must be one of: • `"histone"` = the total histone abundance (default) • `"histone_family"` = the total family abundance • `"histone_group"` = the histone group abundance, this should not be used if target = `"variant"`
`deconv`	A `character(1)` pointing to the rowData column defining the features to deconvolute; for example, hPTMs after alignment mapping.
`sep`	The `character(1)` used to separate different hPTMs/... in the `deconv` column.
`group`	A `character(1)` pointing to the rowData column defining the level at which deconvolution features (e.g., hPTMs) will be grouped; for example, the histone family or histone variant group.

`i`	The index (`integer()`) or name (`character()`) of the assay(s) to be processed.
`name`	Name(s) of the new assay(s) to add to the QFeatures object. Must have the same length as i.
`identifier`	A `character(1)` naming a rowData variable uniquely defining the features.

Value

A `QFeatures` with usage assay(s) added or a usage `SummarizedExperiment` (same as supplied).

References

(1) Demeulemeester N, Gébelin M, Gomes LC, Lingor P, Carapito C, Martens L, Clement L. msqrob2PTM: Differential Abundance and Differential Usage Analysis of MS-Based Proteomics Data at the Posttranslational Modification and Peptidoform Level. Molecular & Cellular Proteomics. 2024 Feb 1;23(2):100708. https://pubs.acs.org/doi/full/10.1021/acs.jproteome.2c00145.

Examples

```
## Not run:
# By default, the entire chromatosome abundance (i.e., aggregated value of
# all histone features) is used as normalization factor
new_qf <- hptm_benchmark |>
  calculateUsage(i = "precursorHistone")
#' # A hPTM will be defined as the unique combination of histone
# group and PTM, e.g., H33K27Me3
rowData(new_qf[["ptm"]])[258, ]
# but if two hPTMs are only quantified from the same features, then these
# will form one "hPTM group".
rowData(new_qf[["ptm"]])[257, ]

# It is also possible to correct for the histone family abundance
new_qf <- new_qf |>
 calculateUsage(i = "precursorHistone", name = "ptm_family", usage_level = "histone_family") |>
# or to do a "variant-corrected" analysis
 calculateUsage(i = "precursorHistone", name = "ptm_group", usage_level = "histone_group")

# The default aggregation function (`MsCoreUtils::robustSummary()`) can be
# slow if feature groups are large, for example when `usage_level = "histone`
# In this case, it can be faster to use `MsCoreUtils::medianPolish()` instead
new_qf_mp <- hptm_benchmark |>
  calculateUsage(
    i = "precursorHistone",
    fun = MsCoreUtils::medianPolish,
    na.rm = TRUE
  )

# Finally, if the unaligned amino acid index is to be used instead
new_qf_unaligned <- hptm_benchmark |>
  calculateUsage(i = "precursorHistone", deconv = "mods_var")
```

```
# Histone variant usage is also easily calculated
new_qf_variant <- hptm_benchmark |>
  calculateUsage(i = "precursorHistone", target = "variant")

## End(Not run)
## Not run:
# By default, the entire chromatosome abundance (i.e., aggregated value of
# all histone features) is used as normalization factor
new_se <- hptm_benchmark[[5]] |>
  calculateUsage()
# A hPTM will be defined as the unique combination of histone
# group and PTM, e.g., H33K27Unmod
rowData(new_se)[258, ]
# but if two hPTMs are only quantified from the same features, then these
# will form one "hPTM group".
rowData(new_se)[257, ]

# It is also possible to correct for the histone family abundance
new_se_hf <- hptm_benchmark[[5]] |>
  calculateUsage(usage_level = "histone_family")
# or to do a "variant-corrected" analysis
new_se_vc <- hptm_benchmark[[5]] |>
  calculateUsage(usage_level = "histone_group")

# The default aggregation function (`MsCoreUtils::robustSummary()`) can be
# slow if feature groups are large, for example when `usage_level = "histone`
# In this case, it can be faster to use `MsCoreUtils::medianPolish()` instead
new_se_mp <- hptm_benchmark[[5]] |>
  calculateUsage(fun = MsCoreUtils::medianPolish, na.rm = TRUE)

# Finally, if the unaligned amino acid index is to be used instead
new_se_unaligned <- hptm_benchmark[[5]] |>
  calculateUsage(deconv = "mods_var")

# Histone variant usage is also easily calculated
new_se_variant <- hptm_benchmark[[5]] |>
  calculateUsage(target = "variant")

## End(Not run)
```

deconvolute *Deconvolute (hPTM) Features*

Description

This is primarily an internal function required before usage aggregation.

Usage

```
deconvolute(
  object,
  ...,
  deconv = "mods_ref",
  sep = ";",
  group = "histone_group"
)

## S4 method for signature 'QFeatures'
deconvolute(
  object,
  i,
  name = "precursorDeconv",
  identifier = "feature_number",
  deconv = "mods_ref",
  sep = ";",
  group = "histone_group"
)

## S4 method for signature 'SummarizedExperiment'
deconvolute(object, deconv = "mods_ref", sep = ";", group = "histone_group")
```

Arguments

`object`	The `QFeatures` or `SummarizedExperiment` object to be deconvoluted.
`...`	Additional arguments passed to specific methods.
`deconv`	A `character(1)` pointing to the rowData column defining the features to deconvolute; for example, hPTMs after alignment mapping.
`sep`	The `character(1)` used to separate different hPTMs/... in the `deconv` column.
`group`	Optional, a `character(1)` pointing to the rowData column defining the level at which deconvolution features (e.g., hPTMs) will be grouped; for example, the histone family or histone variant group.
`i`	The index (`integer()`) or name (`character()`) of the assay(s) to be processed.
`name`	Name(s) of the new assay(s) to add to the QFeatures object. Must have the same length as i.
`identifier`	A `character(1)` naming a rowData variable uniquely defining the features.

Details

One feature can contain multiple hPTMs. Such features will be replicated and assigned one hPTM each. This is needed to be able to use `calculateUsage()`, which relies on `QFeatures::aggregateFeatures()`.

Different hPTMs can have the same underlying information. These "degenerate" PTMs cannot be discerned from one another, so they are added into "hPTM groups".

Value

A `QFeatures` with deconvoluted assay(s) added or a deconvoluted `SummarizedExperiment` (same as supplied).

Examples

```
new_qf <- hptm_benchmark |>
  deconvolute(i = "precursor")
# By default, a hPTM will be defined as the unique combination of histone
# group and PTM, e.g., H33K27Me3
rowData(new_qf[["precursorDeconv"]])[938, ]
# but if two hPTMs are only quantified from the same features, then these
# will form one "hPTM group".
rowData(new_qf[["precursorDeconv"]])[910, ]
# and if the unaligned amino acid index is to be used instead
new_qf_unaligned <- hptm_benchmark |>
  deconvolute(i = "precursor", deconv = "mods_var")
new_se <- hptm_benchmark[[2]] |>
  deconvolute()
# By default, a hPTM will be defined as the unique combination of histone
# group and PTM, e.g., H33K27Me3
rowData(new_se)[938, ]
# but if two hPTMs are only quantified from the same features, then these
# will form one "hPTM group".
rowData(new_se)[910, ]
# and if the unaligned amino acid index is to be used instead
new_se_unaligned <- hptm_benchmark[[2]] |>
  deconvolute(deconv = "mods_var")
```

`generateReport` *Generate Differential Usage Analysis Report*

Description

Dynamically generates a differential usage report using Quarto. All that needs to be supplied is the histone dataset (see `readProgenesis()` and `replaceColData()`), an output directory, and optional settings that modify the analysis. Note that the final QFeatures dataset is invisibly returned so that subsequent custom analysis/plotting is still possible.

Usage

```
generateReport(
  dataset,
  output_dir,
  overwrite = FALSE,
  design_formula = c(factor = "~ 0 + group"),
  random_effects = NULL,
  contrasts = NULL,
  reference_levels = NULL,
  generate_usageplots = c("none", "significant", "all"),
  histone_params = list(),
  msa_params = list(),
  mod_params = list(),
  contaminant_params = list(),
  usage_params = list(),
  ...
)
```

Arguments

`dataset`	The hPTM dataset generated using `readProgenesis()`.
`output_dir`	The path where the final report should be saved (e.g., "./output/").
`overwrite`	If a report with the same name already exists at `output_dir`, should it be overwritten (`logical(1)`?
`design_formula`	A named `character()` of design formula(e), e.g., the default `c(factor = "~ 0 + group")`. The experimental design matrix will only be shown for models that contain "factor" in the name, i.e., the default formula name.
`random_effects`	Optional, a `character()` of categorical variables in the colData that specifies blocking factors for which a random intercept should be fit, e.g., sample preparation batch or study individual.
`contrasts`	A named `list()` of named `character()`s containing relevant contrasts for differential usage testing; for example, `list(factor = c("A vs B" = "groupB - groupA"))` for design_formula `c(factor = "~ 0 + group")` will return a positive LogFC if usage is higher in "conditionB" according to the "factor" model. If no contrasts are provided, differential usage testing will be skipped but the design matrix and QC plots will still be generated for quick finetuning of the analysis.
`reference_levels`	Optional, a named collection of factors (as present in the colData) and their reference level, e.g., `c(group = "wild_type")`. This allows setting the reference group (i.e., intercept) in a "mean-reference" model.
`generate_usageplots`	For which hPTMs/peptidoforms/variants should usageplots be generated? One of "none", "significant" (for any design formula), or "all".
`histone_params`	A `list()` of arguments passed to `histonesFromUniprot()`.
`msa_params`	A `list()` of arguments passed to `alignHistones()` (except for "unaligned_histones").
`mod_params`	A `list()` of arguments passed to `processMods()` (except for "object", "msa", or "i").
`contaminant_params`	A `list()` of arguments passed to `tagContaminants()` (except for "object" or "i").
`usage_params`	A `list()` of arguments passed to `calculateUsage()` (except for "object", "i", "name", or "target").
`...`	Additional parameters passed to `quarto::quarto_render()`.

Details

Under the hood, this function creates a temporary Quarto project directory where a template `.qmd` is then rendered. This template file is dynamically rendered using knitr YAML params that are defined by this function's parameters. The document is rendered as a self-contained HTML, which includes a download button to static assets generated during rendering, i.e., figures and `.csv` assays of relevant usage values.

Value

Invisibly returns the final dataset, including aggregated hPTM assays, for example.

Examples

```
## Not run:
# By default, a "variant-agnostic" usage analysis of human histone samples is performed
# The "ncbtoy" dataset was created by supplying a ".csv" to readProgenesis()
# and then adding metadata using replaceColData()
generateReport(ncbtoy, "./out/")

# Inspecting this HTML report, we can see that hPTMs are not properly parsed
# because they follow a non-standard format. We can use the "*_params" parameters
# to supply this sort of information, see the relevant parameter descriptions
# for more explanation
generateReport(ncbtoy, "./out/", mod_params = list(mod_format = "progenesis_sw"))

# Note that this analysis ends at the design matrix of a simple means model that
# captures the experimental groups. This matrix can help us set up the relevant contrasts
generateReport(
  dataset = ncbtoy,
  output_dir = "./out/",
  mod_params = list(mod_format = "progenesis_sw"),
  contrasts = list(factor = c("A vs B" = "groupcondition_B - groupcondition_A"))
)

# We can of course supply our own design formula(e)
generateReport(
  dataset = ncbtoy,
  output_dir = "./out/",
  mod_params = list(mod_format = "progenesis_sw"),
  design_formula = c(factor = "~ 0 + group", factor_batch = "~ 0 + group + prep_batch"),
  contrasts = list(
    factor = c("A vs B" = "groupcondition_B - groupcondition_A"),
    factor_batch = c(
      "A vs B" = "groupcondition_B - groupcondition_A",
      "batch_B vs batch_A" = "prep_batchB"
    )
  )
)
# or instead supply such variables as a random effect if this makes more sense
generateReport(
  dataset = ncbtoy,
  output_dir = "./out/",
  mod_params = list(mod_format = "progenesis_sw"),
  random_effects = "prep_batch",
  contrasts = list(
    factor = c("A vs B" = "groupcondition_B - groupcondition_A")
  )
)

# Usageplots can be generated if so desired, for example of the significant
# hPTMs/peptidoforms/variants.
# These will be available through a download button at the bottom of the HTML report
# Do note that this can take quite some time/resources to generate all plots
generateReport(
  dataset = ncbtoy,
  output_dir = "./out/",
  mod_params = list(mod_format = "progenesis_sw"),
  contrasts = list(factor = c("A vs B" = "groupcondition_B - groupcondition_A")),
  generate_usageplots = "significant"
)

# Finally, we can easily define the level at which usage is defined
# By default, histone precursor usage and therefore hPTM/histone variant usage
# are defined against the entire chromatosome (i.e., all histone proteins)
# For example, we can choose a "variant-corrected" usage workflow for histone precursors/hPTMs
generateReport(
  dataset = ncbtoy,
  output_dir = "./out/",
  mod_params = list(mod_format = "progenesis_sw"),
  contrasts = list(factor = c("A vs B" = "groupcondition_B - groupcondition_A")),
  usage_params = list(usage_level = "histone_group")
)

## End(Not run)
```

histonesFromUniprot *Retrieve Histone Sequences From UniProt*

Description

Request histone families to the UniProt API and return an AAStringSetList of all requested families.

Usage

```
histonesFromUniprot(
  histone_families = c("H1", "H2A", "H2B", "H3", "H4"),
  query = c("organism_id:9606", "reviewed:true"),
  name_field = "id",
  collapse = " AND "
)
```

Arguments

`histone_families`	A `character()` of histone families to request. One or more of `"H1"`, `"H2A"`, `"H2B"`, `"H3"`, or `"H4"`. Requests all histone families by default.

`query`	A `character()` containing all parts of the target query except the "family" field. For query fields see https://www.uniprot.org/help/query-fields. See also `UniProt.ws::queryUniProt()`.
`name_field`	A `character(1)` specifying the UniProt field to use for naming the sequences. For a full list of possible return fields see https://www.uniprot.org/help/return_fields.
`collapse`	A `character(1)` indicating how to collapse multiple query clauses: either " OR " or " AND ". See also `UniProt.ws::queryUniProt()`.

Value

An `AAStringSetList` where each element corresponds to a histone family and is an `AAStringSet` object of matching histone variant sequences.

Examples

```
## Not run:
# Retrieve all human histone sequences from UniProt (reviewed entries only)
human_histones <- histonesFromUniprot()

# Retrieve only H3 and H4 sequences
h3_h4_histones <- histonesFromUniprot(histone_families = c("H3", "H4"))

# Retrieve mouse histones instead of human (Taxon ID = 10090)
mouse_histones <- histonesFromUniprot(query = c("organism_id:10090", "reviewed:true"))

# Use accession numbers for sequence names instead of entry names
histones_by_accession <- histonesFromUniprot(name_field = "accession")

# Retrieve human histones (reviewed only) and mouse histones (including unreviewed)
human_mouse_histones <- histonesFromUniprot(
  query = c("organism_id:9606 AND reviewed:true", "organism_id:10090"),
  collapse = " OR "
)

## End(Not run)
```

`hptmUsageData` *Get Path to Data File(s)*

Description

Convenience function for accessing data files (located in source at `inst/extdata`) included with the hptmUsage package.

Usage

```
hptmUsageData(file = NULL)
```

Arguments

`file`	Package data file name. If `NULL` (default), all data files are listed.

Examples

```
hptmUsageData()
hptmUsageData("H3.fasta")
```

`hptm_benchmark` *hPTM Benchmark Data*

Description

A preprocessed `QFeatures` of hPTM data, functioning as a benchmark for downstream analysis and statistics. The samples consist of commercial histone extracts treated with a histone deacetylase (HDAC1) in a time-lapse design as described in the reference below.

Usage

```
hptm_benchmark
```

Format

A QFeatures object containing 15 assays of 42 benchmark samples as described in the reference.

assay `precursorRaw` contains the raw (not log-transformed) peptide ion abundances of 677 features

assay `precursor` contains the globally normalized peptide ion abundances of 578 filtered features

assay `precursorCoextr` contains the globally normalized peptide ion abundances of 134 filtered features from co-extracts

assay `co-extracts` contains the robustly summarized protein abundances of 45 co-extracts

assay `precursorHistone` contains the globally normalized peptide ion abundances of 444 filtered features from histones

assay `precursorHistoneNormHistone` contains the usage-normalized (usage defined against the nucleosome) peptide ion abundances of 444 histone features

assay `precursorHistoneNormHistoneDeconv` contains the 1172 features deconvoluted from the assay `precursorHistoneNormHistone`

assay `ptmVariantAgnostic` contains the robustly summarized hPTM usage of 347 histone groups

assay `precursorHistoneNormHistone_family` contains the usage-normalized (usage defined against the corresponding histone family) peptide ion abundances of 444 histone features

assay `precursorHistoneNormHistone_familyDeconv` contains the 1172 features deconvoluted from the assay `precursorHistoneNormHistone_family`

assay `ptmFamilyCorrected` contains the robustly summarized hPTM usage of 347 histone groups

assay `precursorHistoneNormHistone_group` contains the usage-normalized (usage defined against the corresponding histone group) peptide ion abundances of 444 histone features

assay `precursorHistoneNormHistone_groupDeconv` contains the 1172 features deconvoluted from the assay `precursorHistoneNormHistone_group`

assay `ptmVariantCorrected` contains the robustly summarized hPTM usage of 347 histone groups

assay `variant` contains the robustly summarized variant usage of 27 histone variants

rowData contains up to 55 variables, possibly including: (I) variables exported from Progenesis QIP, e.g., "feature_number", "protein", "sequence", "mods", ...; (II) tags exported from Progenesis QIP, e.g., "histone_id_no_err_tol" to "no_protein_id_ra"; (III) variables created during data processing using hptmUsage functions: "histone" = TRUE if the sequence matched any histone variant, else FALSE; "histone_family" = the histone family to which a sequence matched, NA is no match; "core_histone" = TRUE if the sequence matched to one of H2A/H2B/H3/H4, else FALSE; "histone_group" = all histone variants to which the sequence was matched; "start_index" = list of locations of the first amino acid within each matching variant; "end_index" = list of locations of the last amino acid within each matching variant; "ambiguous_match" = TRUE if the sequence has ambiguous histone family assignment or multiple possible positions per variant; "precursor" = precursor string in ProForma notation; "mods_pep" = clean mod string with location indexes on the peptide sequence; "mods_var" = clean mod string with location indexes on all matching histone variants; "mods_msa" = clean mod string with location indexes on all matching histone variants after MSA; "mods_ref" = clean mod string with "consensus" location index(es) on the histone family reference sequence after MSA; "contaminant" = TRUE if identified as a potential contaminant using hptmUsage::tagContaminants() else FALSE; "nNA" = the number of samples in which the precursor was missing; "pNA" = the fraction of samples in which the precursor was missing; ".n" = the number of precursors from which the abundance/usage was summarized; "hptm" = combination of histone variant + amino acid + location + PTM, e.g., H33_BOVIN#K|27|Me3, possibly in a "degenerated" hPTM group separated by ";".

colData contains 7 columns of sample metadata, including "group", "time", and "treated" (experimental factors), and "include"/"outlier" (tags for analysis)

Source

https://www.ebi.ac.uk/pride/archive/projects/PXD009910

References

Clerck, L. D.; Willems, S.; Daled, S.; Puyvelde, B. V.; Verhelst, S.; Corveleyn, L.; Deforce, D.; Dhaenens, M. An Experimental Design to Extract More Information from MS-Based Histone Studies. Molecular Omics 2021, 17 (6), 929–938. https://doi.org/10.1039/D1MO00201E.

Examples

```
hptm_benchmark |>
  show()
```

matchHistones *Match Sequences to Histone Variants*

Description

Adds columns to the `rowData` of the given `QFeatures` or `SummarizedExperiment`, including checking for common errors.

Usage

```
matchHistones(object, matching_subject, ..., sequence_col = "sequence")

## S4 method for signature 'QFeatures,AAStringSetList'
matchHistones(
  object,
  matching_subject,
  i,
  progress = TRUE,
  drop_ambiguous = TRUE,
  sequence_col = "sequence"
)

## S4 method for signature 'SummarizedExperiment,AAStringSetList'
matchHistones(
  object,
  matching_subject,
  progress = TRUE,
  drop_ambiguous = TRUE,
  sequence_col = "sequence"
)
```

Arguments

`object`	The `QFeatures` or `SummarizedExperiment` object in which the sequences should be matched to histone variants.
`matching_subject`	An `AAStringSetList` of histone families and corresponding sequences. For example, the "unaligned" element resulting from `alignHistones()`.
`...`	Additional arguments passed to specific methods.
`sequence_col`	The column name (`character(1)`) containing the sequences.
`i`	The index (`integer()`) or name (`character()`) of the assay(s) to be processed.
`progress`	Show a progress bar? Defaults to `TRUE`.
`drop_ambiguous`	Drop family- or position-ambiguous features? Defaults to `TRUE`.

Value

A `QFeatures` or `SummarizedExperiment` (same as supplied) with modified rowData of histone variant matches and the location.

Examples

```
## Not run:
ncbtoy |>
  matchHistones(aligned_histones$unaligned, 1)

## End(Not run)
## Not run:
ncbtoy[[1]] |>
  matchHistones(aligned_histones$unaligned)

## End(Not run)
```

`ncbtoy` *Example hPTM Data*

Description

A minimally preprocessed `QFeatures` of hPTM data for testing and demonstration purposes. The samples consist of bottom-up histone extracts from naive (condition_A) and primed (condition_B) stem cell cultures as described in the reference below.

Usage

```
ncbtoy
```

Format

A QFeatures object with 1 assay named `precursorRaw`. This assay contains 5663 features (peptide ions) and 10 samples from two conditions ('condition_A' and 'condition_B').

assay contains the raw peptide ion abundances

rowData contains 21 variables, including (most relevant) "protein", "sequence", and "mods"

colData contains 8 columns of sample metadata, including the "group" (biological factor), "ms_run"/"date_collected"/"prep_ (technical factors), and "include"/"outlier" (tags for analysis)

Source

https://www.ebi.ac.uk/pride/archive/projects/PXD028162

References

Zijlmans, D. W.; Talon, I.; Verhelst, S.; Bendall, A.; Van Nerum, K.; Javali, A.; Malcolm, A. A.; Van Knippenberg, S. S. F. A.; Biggins, L.; To, S. K.; Janiszewski, A.; Admiraal, D.; Knops, R.; Corthout, N.; Balaton, B. P.; Georgolopoulos, G.; Panda, A.; Bhanu, N. V.; Collier, A. J.; Fabian, C.; Allsop, R. N.; Chappell, J.; Pham, T. X. A.; Oberhuemer, M.; Ertekin, C.; Vanheer, L.; Athanasouli, P.; Lluis, F.; Deforce, D.; Jansen, J. H.; Garcia, B. A.; Vermeulen, M.; Rivron, N.; Dhaenens, M.; Marks, H.; Rugg-Gunn, P. J.; Pasque, V. Integrated Multi-Omics Reveal Polycomb Repressive Complex 2 Restricts Human Trophoblast Induction. Nat Cell Biol 2022, 24 (6), 858–871. https://doi.org/10.1038/s41556-022-00932-w.

Examples

```
ncbtoy |>
  show()
```

---

normalizeUsage *Usage Normalization of hPTMs*

Description

This is primarily an internal function required before usage aggregation.

Usage

```
normalizeUsage(
  object,
  ...,
  usage_level = c("histone", "histone_family", "histone_group")
)

## S4 method for signature 'QFeatures'
normalizeUsage(
  object,
  i,
  name = "precursorNorm",
  ...,
  usage_level = c("histone", "histone_family", "histone_group")
)

## S4 method for signature 'SummarizedExperiment'
normalizeUsage(
  object,
  ...,
  usage_level = c("histone", "histone_family", "histone_group")
)
```

Arguments

`object`	The `QFeatures` or `SummarizedExperiment` object to be usage normalized.
`...`	Additional arguments passed to `QFeatures::aggregateFeatures()`.
`usage_level`	A `character(1)` defining the level at which usage will be calculated, i.e., what is considered the "parent protein". Must be one of: • `"histone"` = the total histone abundance (default) • `"histone_family"` = the total family abundance • `"histone_group"` = the histone group abundance
`i`	The index (`integer()`) or name (`character()`) of the assay(s) to be processed.
`name`	Name(s) of the new assay(s) to add to the QFeatures object. Must have the same length as i.

Details

Histone precursors are normalized by subtracting the parent protein abundance from the (globally normalized) feature abundances. See also Demeulemeester et al. (1).

Value

A `QFeatures` with usage normalized assay(s) added or a deconvoluted `SummarizedExperiment` (same as supplied).

References

(1) Demeulemeester N, Gébelin M, Gomes LC, Lingor P, Carapito C, Martens L, Clement L. msqrob2PTM: Differential Abundance and Differential Usage Analysis of MS-Based Proteomics Data at the Posttranslational Modification and Peptidoform Level. Molecular & Cellular Proteomics. 2024 Feb 1;23(2):100708. https://pubs.acs.org/doi/full/10.1021/acs.jproteome.2c00145.

Examples

```
## Not run:
# By default, the entire chromatosome abundance (i.e., aggregated value of
# all histone features) is used as normalization factor
new_qf <- hptm_benchmark |>
  normalizeUsage(i = "precursorHistone")
# It is also possible to to correct for the histone family abundance
new_qf_hf <- hptm_benchmark |>
  normalizeUsage(i = "precursorHistone", usage_level = "histone_family")
# or to do a "variant-corrected" analysis
new_qf_vc <- hptm_benchmark |>
  normalizeUsage(i = "precursorHistone", usage_level = "histone_group")

# The default aggregation function (`MsCoreUtils::robustSummary()`) can be
# slow if feature groups are large, for example when `usage_level = "histone`
# In this case, it can be faster to use `MsCoreUtils::medianPolish()` instead
new_qf_mp <- hptm_benchmark |>
```

```
  normalizeUsage(
    i = "precursorHistone",
    fun = MsCoreUtils::medianPolish,
    na.rm = TRUE
  )

## End(Not run)
## Not run:
# By default, the entire chromatosome abundance (i.e., aggregated value of
# all histone features) is used as normalization factor
new_se <- hptm_benchmark[[5]] |>
  normalizeUsage()
# It is also possible to to correct for the histone family abundance
new_se_hf <- hptm_benchmark[[5]] |>
  normalizeUsage(usage_level = "histone_family")
# or to do a "variant-corrected" analysis
new_se_vc <- hptm_benchmark[[5]] |>
  normalizeUsage(usage_level = "histone_group")

# The default aggregation function (`MsCoreUtils::robustSummary()`) can be
# slow if feature groups are large, for example when `usage_level = "histone`
# In this case, it can be faster to use `MsCoreUtils::medianPolish()` instead
new_se_mp <- hptm_benchmark[[5]] |>
  normalizeUsage(fun = MsCoreUtils::medianPolish, na.rm = TRUE)

## End(Not run)
```

processMods *Process Histone Modifications*

Description

hPTMs are rewritten in a common format ("aa|loc|mod;..."), for example "K|27|Me3;K|36|Ac;K|37|Unmod". This involves mapping the hPTM location from within a peptide to the matching histone variant(s). It also allows to add "Unmod" pseudo-hPTMs, remove uninformative hPTMs (e.g., propionylation from chemical derivatization), and rename hPTMs (e.g., from "Trimethyl" to "Me3").

Usage

```
processMods(
  object,
  msa,
  ...,
  mod_format = c("progenesis", "progenesis_sw"),
  unmods = c("K"),
  strip_mods = list(Propionyl = c("K", "N-term")),
  rename_mods = NULL
)
```

```
## S4 method for signature 'QFeatures,list'
processMods(
  object,
  msa,
  i,
  mod_format = c("progenesis", "progenesis_sw"),
  unmods = c("K"),
  strip_mods = list(Propionyl = c("K", "N-term")),
  rename_mods = NULL
)

## S4 method for signature 'SummarizedExperiment,list'
processMods(
  object,
  msa,
  mod_format = c("progenesis", "progenesis_sw"),
  unmods = c("K"),
  strip_mods = list(Propionyl = c("K", "N-term")),
  rename_mods = NULL
)
```

Arguments

`object`	The `QFeatures` or `SummarizedExperiment` object in which the mods should be processed.
`msa`	A `list()` of `AAStringSetList` objects containing unaligned, aligned, and optionally reference histone sequences. Each `AAStringSetList` element contains sequences of one histone family. For example, the (default) result from `alignHistones()`.
`...`	Additional arguments passed to specific methods.
`mod_format`	The format of the modification string. Defaults to `"progenesis"`. Must be one of: • `"progenesis"` = "[loc] mod (aa)|..." • `"progenesis_sw"` = "[loc] (aa) mod|..."
`unmods`	A `character()` of amino acids to which "Unmod" pseudo-hPTMs should be added. Optional, `NULL` disables adding unmods.
`strip_mods`	A named `list(mod1 = character("aa1", ...), ...)` of modifications to strip. Optional, `NULL` disables stripping mods.
`rename_mods`	A `list()` of two-sided formulas where the LHS is the old name and the RHS the new name (e.g., `"old" ~ "new"`) specifying how to rename hPTMs. Optional.
`i`	The index (`integer()`) or name (`character()`) of the assay(s) to be processed.

Value

A `QFeatures` or `SummarizedExperiment` (same as supplied) with the rowData containing four new columns, where each column differs in the locations of hPTMs:

- "mods_pep" = location within peptide
- "mods_var" = location within histone variant(s)
- "mods_msa" = location within histone variant(s) after multiple sequence alignment (see `alignHistones()`)
- "mods_ref" = corresponding location within the aligned reference sequence (see `alignHistones()`), only when `msa` contains "msa_ref"

One additional column is added: "precursor", which is the ProForma combination of sequence, mods, and charge.

Examples

```
## Not run:
qf <- matchHistones(ncbtoy, aligned_histones$unaligned, 1)

# in this dataset, mods follow the "progenesis_sw" format
processMods(qf, aligned_histones, 1, mod_format = "progenesis_sw")

# "Unmod" can be added to other amino acids and we can remove chemical artifacts
processMods(
  qf,
  aligned_histones,
  1,
  mod_format = "progenesis_sw",
  unmods = c("K", "R"),
  strip_mods = list(Fo = c("K"))
)

# maybe we would like to write out hPTMs in full
processMods(
  qf,
  aligned_histones,
  1,
  mod_format = "progenesis_sw",
  rename_mods = list("Ac" ~ "Acetyl", "Me2" ~ "Dimethyl", "Me3" ~ "Trimethyl")
)

## End(Not run)
## Not run:
se <- matchHistones(ncbtoy[[1]], aligned_histones$unaligned)

# in this dataset, mods follow the "progenesis_sw" format
processMods(se, aligned_histones, mod_format = "progenesis_sw")

# "Unmod" can be added to other amino acids and we can remove chemical artifacts
processMods(
  se,
  aligned_histones,
  mod_format = "progenesis_sw",
  unmods = c("K", "R"),
  strip_mods = list(Fo = c("K"))
)
```

```
# maybe we prefer to write out hPTMs in full
processMods(
  se,
  aligned_histones,
  mod_format = "progenesis_sw",
  rename_mods = list("Ac" ~ "Acetyl", "Me2" ~ "Dimethyl", "Me3" ~ "Trimethyl")
)

## End(Not run)
```

---

readProgenesis *Read a Progenesis QIP Peptide Ion Data Export*

Description

A Progenesis QIP peptide ion data `.csv` file is cleaned up to make it compatible with the `QFeatures` framework. This involves subsetting the quantitative data to one type, e.g., raw abundances, sanitizing sample names, generating initial metadata from the Progenesis experimental design, and handling duplicated features.

Usage

```
readProgenesis(
  file,
  quant = "Raw abundance",
  generate_metadata = FALSE,
  overwrite_metadata = FALSE,
  simplify_column_names = TRUE,
  duplicates = "max"
)
```

Arguments

`file`	The path (`character(1)`) to the Progenesis QIP peptide ion data `.csv` file. This file should contain all identified features, including co-extracts, and at least the properties of feature number, charge, protein, sequence, variable modifications, and a quantitative value (see `quant`).
`quant`	The quantitative value to use: `"Raw abundance"` (default), `"Normalized abundance"`, or `"Intensity"`.
`generate_metadata`	Generate a metadata `.csv` file that can be manually edited (default `FALSE`, see also `overwrite_metadata`)? Can also be a path to the location where the metadata file will be written (e.g., `"./data/metadata.csv"`). Updated metadata can later be added to a `QFeatures` object using `replaceColData()`.
`overwrite_metadata`	Overwrite the metadata file if it already exists (default `FALSE`)?

`simplify_column_names`	Remove pre-/suffixes from sample names (default `TRUE`)?
`duplicates`	How to handle duplicated features (same sequence, charge, and modifications). Can be `"max"` (default, keep feature with highest mean abundance across samples), `"sum"` (sum abundances, note that this only keeps feature properties of the first encountered duplicate), or `NULL` (do nothing).

Value

A `QFeatures` containing the Progenesis QIP dataset.

See Also

The `QFeatures::QFeatures()` class to read about how to manipulate the resulting `QFeatures` object (if `return_qfeatures` was `TRUE`).

Examples

```
# Simplest case
readProgenesis("./all_ion_export.csv")

# Generate a metadata `.csv` file that can be manually edited
readProgenesis("./all_ion_export.csv", generate_metadata = TRUE)
# A path can also be supplied, instead of having the file generated next to
# the `.csv` source.
readProgenesis("./all_ion_export.csv", generate_metadata = "./metadata.csv")
# Note, if the metadata file already exists, nothing will be written unless
# `overwrite_metadata` is set to `TRUE`

# Use feature intensities instead of raw abundances
readProgenesis("./all_ion_export.csv", quant = "Intensity")

# Do not simplify (remove pre- and suffixes) sample names
readProgenesis("./all_ion_export.csv", simplify_column_names = FALSE)

# Handle duplicated features by summing their abundances
readProgenesis("./all_ion_export.csv", duplicates = "sum")
```

`replaceColData` *Safely Replace* `colData`

Description

Replaces `colData` of the given `QFeatures` or `SummarizedExperiment`, including checking for common errors.

Usage

```
replaceColData(object, colData, ...)

## S4 method for signature 'QFeatures_OR_SummarizedExperiment,character'
replaceColData(object, colData, custom_coltypes = NULL)

## S4 method for signature 'QFeatures_OR_SummarizedExperiment,tbl_df'
replaceColData(object, colData)
```

Arguments

`object`	The `QFeatures` or `SummarizedExperiment` object of which the `colData` should be replaced.
`colData`	New colData to be added to `object`. Can be either a path (`character(1)`) pointing to a colData `.csv` file, for example generated using `readProgenesis()`, or a `tbl_df` where rows are samples and columns are properties of the samples. The samples need to match those in the `object` and the columns should contain at least the "quantCols", "original_name", "group", "include", and "outlier".
`...`	Additional arguments passed to specific methods.
`custom_coltypes`	A named `list(...)` of types for non-standard columns, defaulting to `readr::col_factor()`. See `readr::cols()` for more information. There is no need to specify the types for "quantCols", "original_name", "group", "include", and "outlier".

Value

A `QFeatures` or `SummarizedExperiment` (same as supplied) with the new colData.

Examples

```
replaceColData(
  ncbtoy,
  hptmUsageData("ncbtoy_coldata.csv"),
  list(
    ms_run = readr::col_integer(),
    date_collected = readr::col_date(format = "%y%m%d"),
    prep_batch = readr::col_factor()
  )
)
readr::read_csv(hptmUsageData("ncbtoy_coldata.csv")) |>
  dplyr::mutate(group = as.factor(group)) |>
  replaceColData(
    ncbtoy,
    colData = _
  )
```

`tagContaminants` *Tag Contaminant Proteins*

Description

Use custom contaminant libraries from Frankenfield et al. (1), where histone entries (Cont_A1A4R1, Cont_P62803, Cont_Q64475) were manually removed, to tag contaminant proteins for later filtering. See `https://github.com/HaoGroup-ProtContLib/Protein-Contaminant-Libraries-for-DDA-and-DIA-Protec` for more information.

Usage

```
tagContaminants(
  object,
  ...,
  library = c("universal", "cell_culture", "mouse_tissue", "rat_tissue",
    "neuron_culture", "stem_cell_culture"),
  sequence = "sequence"
)

## S4 method for signature 'QFeatures'
tagContaminants(
  object,
  i,
  library = c("universal", "cell_culture", "mouse_tissue", "rat_tissue",
    "neuron_culture", "stem_cell_culture"),
  sequence = "sequence"
)

## S4 method for signature 'SummarizedExperiment'
tagContaminants(
  object,
  library = c("universal", "cell_culture", "mouse_tissue", "rat_tissue",
    "neuron_culture", "stem_cell_culture"),
  sequence = "sequence"
)
```

Arguments

`object`	The `QFeatures` or `SummarizedExperiment` object in which the contaminants should be tagged.
`...`	Additional arguments passed to specific methods.
`library`	The name (`character(1)`) of the contaminant library to use. Must be one of: • `"universal"` (default) • `"cell_culture"` • `"mouse_tissue"`

- `"rat_tissue"`
- `"neuron_culture"`
- `"stem_cell_culture"`

`sequence`	The name (`character(1)`) of the rowData column containing (stripped) peptide sequences.
`i`	The index (`integer()`) or name (`character()`) of the assay(s) to be processed.

Value

A `QFeatures` or `SummarizedExperiment` (same as supplied) with the rowData containing one new `logical()` column "contaminant".

References

(1) Frankenfield AM, Ni J, Ahmed M, Hao L. Protein contaminants matter: building universal protein contaminant libraries for DDA and DIA proteomics. Journal of proteome research. 2022 Jul 6;21(9):2104-13. https://pubs.acs.org/doi/full/10.1021/acs.jproteome.2c00145.

Examples

```
# By default, contaminants will be tagged from the "universal" library
qf <- ncbtoy |>
  tagContaminants(i = "precursorRaw")
rowData(qf)
# but we can easily specify other libraries
qf_alt <- ncbtoy |>
  tagContaminants(i = "precursorRaw", library = "stem_cell_culture")
rowData(qf_alt)
# By default, contaminants will be tagged from the "universal" library
se <- ncbtoy[[1]] |>
  tagContaminants()
rowData(se)
# but we can easily specify other libraries
se_alt <- ncbtoy[[1]] |>
  tagContaminants(library = "stem_cell_culture")
rowData(se_alt)
```

`usagePlot` *Generate Usage Plots*

Description

This function generates line plots showing the usage of histone variants, precursors, or PTMs across samples. It visualizes the relevant usage normalization factors, corrected usages, and model estimates.

Usage

```
usagePlot(object, assays, model, significant_only = TRUE, show_legend = TRUE)
```

Arguments

`object`	A `QFeatures` object containing the histone data, including models from `msqrob2::msqrob()` etc.
`assays`	A named `list()` of assays (`character()` or `integer()`) to plot (hPTM and/or variant group level), where names should correspond to the assay level, i.e., "ptm" or "variant". For example `list(ptm = "ptm", variant = "variant"),`.
`model`	A named `character()` of formula(e) that were fit using `msqrob2::msqrob()`. Used to reconstruct model estimates. For example, `c(meansReference = "~ group", regressionTreat = "~ cell_line + treatment")`.
`significant_only`	Logical. Whether to generate lineplots for all features or only those that weres significant in **any** contrasts of the supplied model(s).
`show_legend`	Logical. Whether to show the legend.

Value

A list of `ggplot` objects, nested by assay level and assay name.

Index